ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю»

Выполнил работу

Воробьев Егор

Академическая группа J3113

Принято

Должность, Дунаев Максим

Санкт-Петербург

2024

1. **Введение**

Цель: Найти подмассивы размера 5, в которых сумма элементов равна нулю.

Задачи:

1) Написать комбинаторный код за O(N^5)

2) Протестировать и сравнить работу кода на различных размерах массива

1. **Теоретическая подготовка**

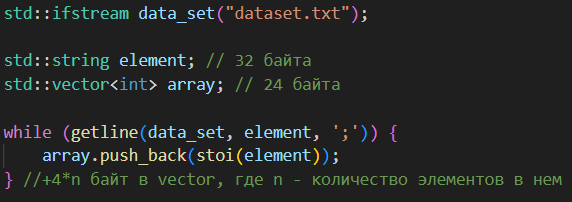
Для написания комбинаторного кода воспользуемся вложенными циклами. Чтобы сложность алгоритма была O(N^5), используем 5 циклов, вложенных друг в друга (кроме первого).

Типы данных:

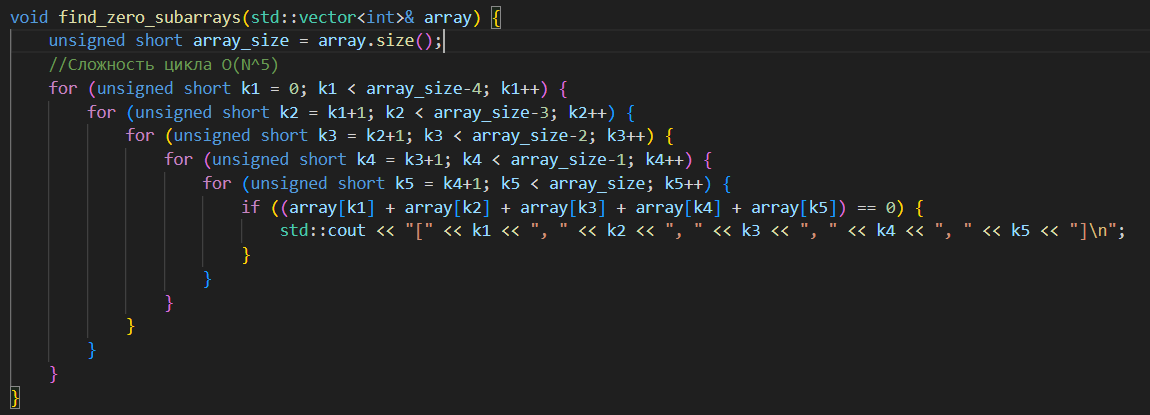
1. std::vector – для реализации основого массива, в котором будем находить подмассивы;
2. std::string – для считывания входных данных из файла поэлементно.

Заголовочные файлы:

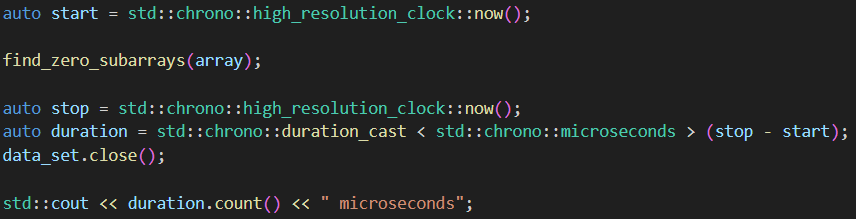
1. <vector> - для использования типа данных std::vector;
2. <string> - для использования типа данных std::string;
3. <fstream> - для считывания входных данных из файла;
4. <iostream> - для вывода подмассивов;
5. <chrono> - для измерения времени выполнения алгоритма.
6. **Реализация**
7. Считывание входных данных из файла “dataset.txt” с помощью функции getline, которая на каждой итерации цикла while записывает элементы в переменную element типа std::string до знака “;”, и массива array типа std::vector, в который на каждой итерации добавляется ранее полученный элемент, конвертированный из типа std::string в тип int с помощью функции stoi. Цикл while работает до тех пор, пока функция getline может считывать данные.

****

1. Основной частью кода является функция find\_zero\_subarrays, принимающая на вход указазатель на массив, в котором необходимо найти подмассивы длины 5. Решение заключается в полном переборе всех возможных наборов из 5 чисел. Для этого используется 5 циклов, в каждом вложенном цикле начальная переменная-счетчик увеличивается на 1 по сравнению с прошлым циклом, это сделано для того, чтобы не рассматривать одни и те же элементы. Также в каждом цикле конечное значение переменной-счетчика увеличивается на 1, начиная с array\_size-4, это необходимо для того, чтобы не выйти за пределы массива. После итераций каждого цикла имеем 5 индексов элементов массива, вычисляем их сумму, если она равна 0, то выводим найденный подмассив.



1. Далее измеряем время выполнения функции, рассмотренной в прошлом пункте. Перед вызовом функции фиксируем стартовое время с помощью функции std::chrono::high\_resolution\_clock::now, которая возвращает текущее время. Затем вызываем функцию find\_zero\_subarrays и после ее выполнения еще раз фиксируем время. Теперь вычисляем само время выполнения в микросекундах и выводим его.



1. **Экспериментальная часть**

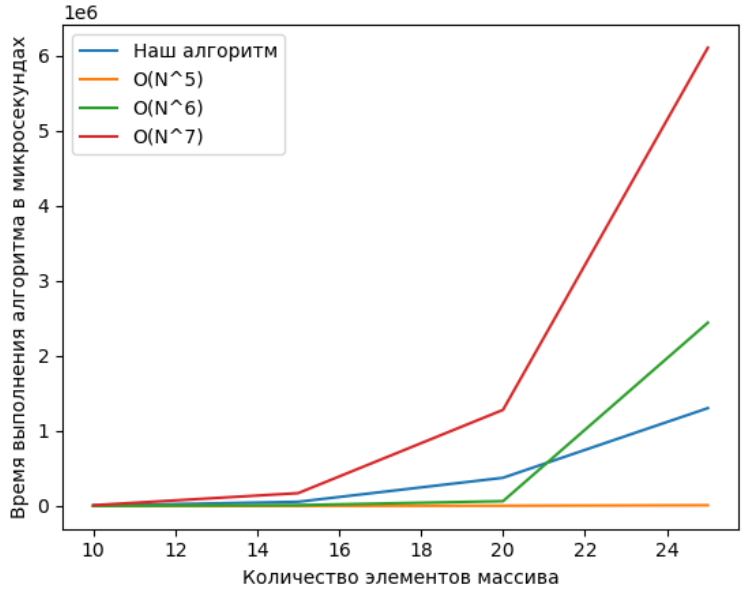
Теоретически, сложность алгоритма **≈** O(N^5).

Для определения эффективности алгоритма, было измерено его время выполнения на разных размерах массива. Собранная статистика отображена в таблице №1.

Таблица №1 – Время выполнения алгоритма при различных размерах массива

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Размер массива 10 | Размер массива 15 | Размер массива 20 | Размер массива 25 |
| Попытка №1 | 3550 | 45368 | 336381 | 1075877 |
| Попытка №2 | 3693 | 48291 | 376242 | 1208896 |
| Попытка №3 | 3899 | 83409 | 411375 | 1096039 |
| Попытка №4 | 4280 | 56450 | 369069 | 1178468 |
| Попытка №5 | 5635 | 44923 | 384835 | 1957319 |
| Среднее | 4211.4 | 55688.2 | 375580.4 | 1303319.8 |
| O(N^5) | 100 | 759.4 | 3200 | 9765.6 |
| O(N^6) | 1000 | 11390.6 | 64000 | 2441140.6 |
| O(N^7) | 10000 | 170859.4 | 1280000 | 6103515.6 |

Ниже предоставлен график, визуализирующий данные из таблицы №1.

Изображение №1 – График зависимости времени выполенения алгоритма от размера массива и нотаций, близких к нашей (время выбрано среднее)

Как видно из графика, сложность алгоритма точно больше O(N^5), в некоторый момент больше и O(N^6), но при N > 21 график O(N^6) возрастает сильнее, чем наш алгоритм. Поэтому, на самом деле, ситуация следующая: O(N^5) < сложность алгоритма < O(N^6). Полученный результат можно объяснить частым выводом std::cout и проверкой условия на нулевую сумму. Эти действия так же занимают определенное время, из-за чего сложность получилась больше ожидаемой.

1. **Заключение**

В ходе выполнения работы мною был реализован алгоритм поиска подмассивов длины 5, сумма которых равна 0. Цель работы была достигнута путём тестирования алгоритма на разных входных данных. Полученные результаты примерно равны теоретическим оценкам сложности алгоритма.

1. **Приложения**

ПРИЛОЖЕНИЕ 1

Полный код алгоритма

